Introducción a la Criptografía Moderna

Funciones de Hash

Rodrigo Abarzúa[†],

[†] Universidad de Santiago de Chile
rodrigo.abarzua@usach.cl

June 3, 2014

Funciones de Hash

- Las funciones hash son una importante primitiva criptográfica y son ampliamente utilizados en protocolos.
- Ellos calculan un resumen de un mensaje (o una compresión del mensaje) que es una cadena de bits corta de longitud fija.
- En un mensaje en particular, el resumen del mensaje, o el valor hash, pueden ser vistos como la huella digital de un mensaje, es decir, una representación única de un mensaje.
- A diferencia de otros algoritmos de cifrado, las funciones hash no tienen una clave.
- El uso de funciones hash de criptografía es múltiple:
  - Las funciones hash son una parte esencial de los sistemas de firma digital (como se explica en las clases anteriores) y de los códigos de autenticación de mensajes (MAC) (que estudiaremos más adelante).
  - Las funciones hash son utilizados para el almacenamiento de hashes de contraseñas o claves de derivación.

En esta sección estudiaremos:

- Por qué son necesarias las funciones de hash en los esquemas de firma digital.
- Propiedades importantes de las funciones hash.
- Un análisis de la seguridad de las funciones hash, incluyendo una introducción al ataque utilizando la paradoja del cumpleaños.
- Como funciona la función hash SHA-1.

Motivación: Firma de los mensajes largos

- A pesar de que las funciones hash tienen muchas aplicaciones en criptografía moderna, son quizás más conocido por el importante papel que desempeñan en la práctica el uso de la firma digital.
- En clases anteriores, hemos introducido esquemas de firma basada en el algoritmo RSA, el problema del logaritmo discreto en puntos de curvas elípticas.
- Para todos los esquemas, la longitud del plaintext es acotada.
- Por ejemplo, en el caso de RSA, el mensaje no puede ser mayor que el módulo, que es en la práctica a menudo entre 1024 y 3072-bits de longitud.
- Recuerde que esto se traduce en sólo 128 a 384-bytes. Pero la mayoría de los correos electrónicos son mucho más largo que eso.
- Hasta el momento, hemos dejado de lado el hecho de que en la práctica el plaintext x a menudo será mucho más grande que estos tamaños.
- La pregunta que surge en este punto es:

¿Cómo vamos a calcular de manera eficiente firmas de mensajes de gran tamaño?

Enfoque Divida el Mensaje

Un enfoque intuitivo sería similar al modo de cifrados de bloque:

Divida el mensaje x en x_i en bloques de tamaño menor que el tamaño permitido de entrada del algoritmo de firma, y firme cada bloque por separado, tal como se representa en la siguiente figura.

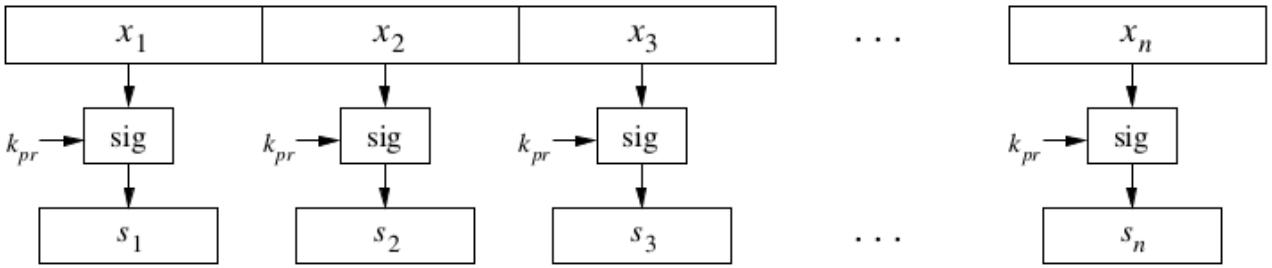


Figure: Enfoque inseguro a la firma de los mensajes largos

Enfoque Divida el Mensaje

Problema 1: Alta carga computacional

1. Las firmas digitales se basan en operaciones de cálculo en criptosistemas asimétricos como exponenciales modulares de números enteros grandes

$$\beta \equiv \alpha^d \bmod p.$$

2. Incluso si una sola operación consume una pequeña cantidad de tiempo (y energía, lo que es relevante en aplicaciones móviles), las firmas de mensajes de gran tamaño, por ejemplo, archivos adjuntos de correo electrónico o archivos multimedia, tomaría demasiado tiempo en las computadoras actuales.
3. Además, no sólo el firmante tiene que calcular la firma, sino que el verificador también tiene que gastar una cantidad similar de tiempo y energía para verificar la firma.

Enfoque Divida el Mensaje

Problema 2: Overhead Mensaje

1. Este enfoque duplica la sobrecarga mensaje porque no sólo debe ser enviado el mensaje, sino también la firma, que es de la misma longitud en este caso.
2. Por ejemplo, un archivo de 1-MB debe resultar en una firma de RSA de longitud de 1-MB, por lo que un total de 2-MB se debe transmitir.

Problema 3: Limitaciones de seguridad

Enfoque Divida el Mensaje

1. Este es el problema más grave de este enfoque, ya que para firmar un mensaje largo x, al firmar una secuencia de bloques de mensajes x_i individualmente. Conduce inmediatamente a nuevos ataques:
   - Por ejemplo, un atacante podría eliminar mensajes individuales y las firmas correspondientes,
   - o podría reordenar los mensajes y firmas,
   - o podría volver a montar nuevos mensajes y firmas de fragmentos de mensajes anteriores y firmas, etc.
   - Incluso aunque un atacante no pueden realizar manipulaciones en un bloque individual, no tenemos protección para todo el mensaje.

Problema 3: Limitaciones de seguridad

Enfoque Divida el Mensaje

1. Este es el problema más grave de este enfoque, ya que para firmar un mensaje largo x, al firmar una secuencia de bloques de mensajes x_i individualmente. Conduce inmediatamente a nuevos ataques:
   - Por ejemplo, un atacante podría eliminar mensajes individuales y las firmas correspondientes,
   - o podría reordenar los mensajes y firmas,
   - o podría volver a montar nuevos mensajes y firmas de fragmentos de mensajes anteriores y firmas, etc.
   - Incluso aunque un atacante no pueden realizar manipulaciones en un bloque individual, no tenemos protección para todo el mensaje.

Problema 3: Limitaciones de seguridad

Enfoque Divida el Mensaje

1. Este es el problema más grave de este enfoque, ya que para firmar un mensaje largo x, al firmar una secuencia de bloques de mensajes x_i individualmente. Conduce inmediatamente a nuevos ataques:
   - Por ejemplo, un atacante podría eliminar mensajes individuales y las firmas correspondientes,
   - o podría reordenar los mensajes y firmas,
   - o podría volver a montar nuevos mensajes y firmas de fragmentos de mensajes anteriores y firmas, etc.
   - Incluso aunque un atacante no pueden realizar manipulaciones en un bloque individual, no tenemos protección para todo el mensaje.

Problema 3: Limitaciones de seguridad

Enfoque Divida el Mensaje

1. Este es el problema más grave de este enfoque, ya que para firmar un mensaje largo x, al firmar una secuencia de bloques de mensajes x_i individualmente. Conduce inmediatamente a nuevos ataques:
   - Por ejemplo, un atacante podría eliminar mensajes individuales y las firmas correspondientes,
   - o podría reordenar los mensajes y firmas,
   - o podría volver a montar nuevos mensajes y firmas de fragmentos de mensajes anteriores y firmas, etc.
   - Incluso aunque un atacante no pueden realizar manipulaciones en un bloque individual, no tenemos protección para todo el mensaje.

Problemas:

- Por lo tanto, para el desempeño así como por razones de seguridad, nos gustaría tener una firma de longitud relativamente pequeña aplicable a mensaje de longitud arbitraria.
- La solución a este problema son las "Funciones Hash".
- Si tuviéramos una función hash para calcular una firma digital del mensaje x, donde se podría realizar la operación de firma, como se muestra en la siguiente figura:

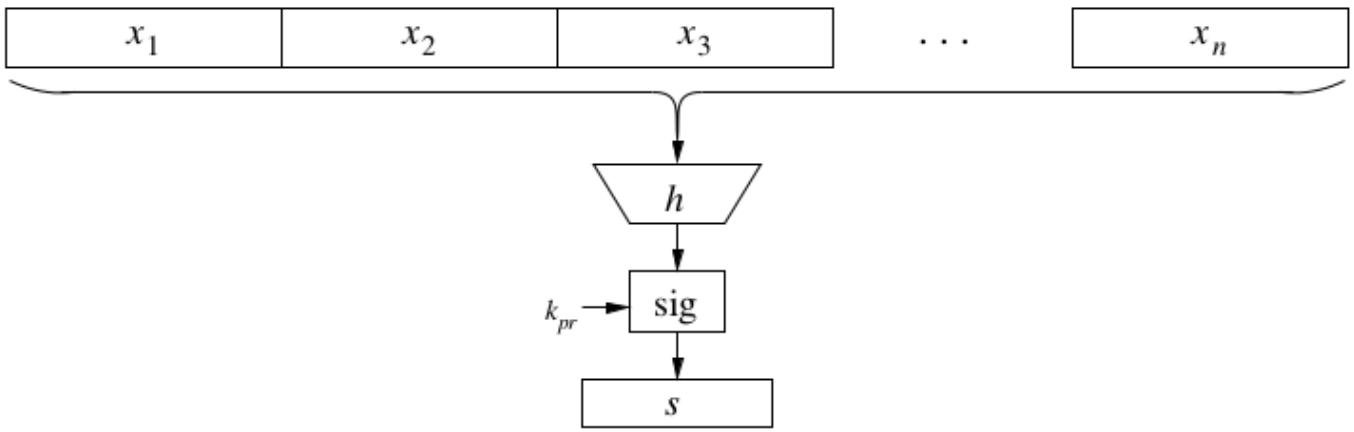


Figure: Firma de los mensajes largos con una función hash

Basic Protocol for Digital Signature with a Hash Function

Supongamos que tenemos tal función hash, entonces un protocolo básico para un esquema de firma digital con una función hash. Podría ser el siguiente:

- Bob quiere enviar un mensaje firmado digitalmente para Alice.

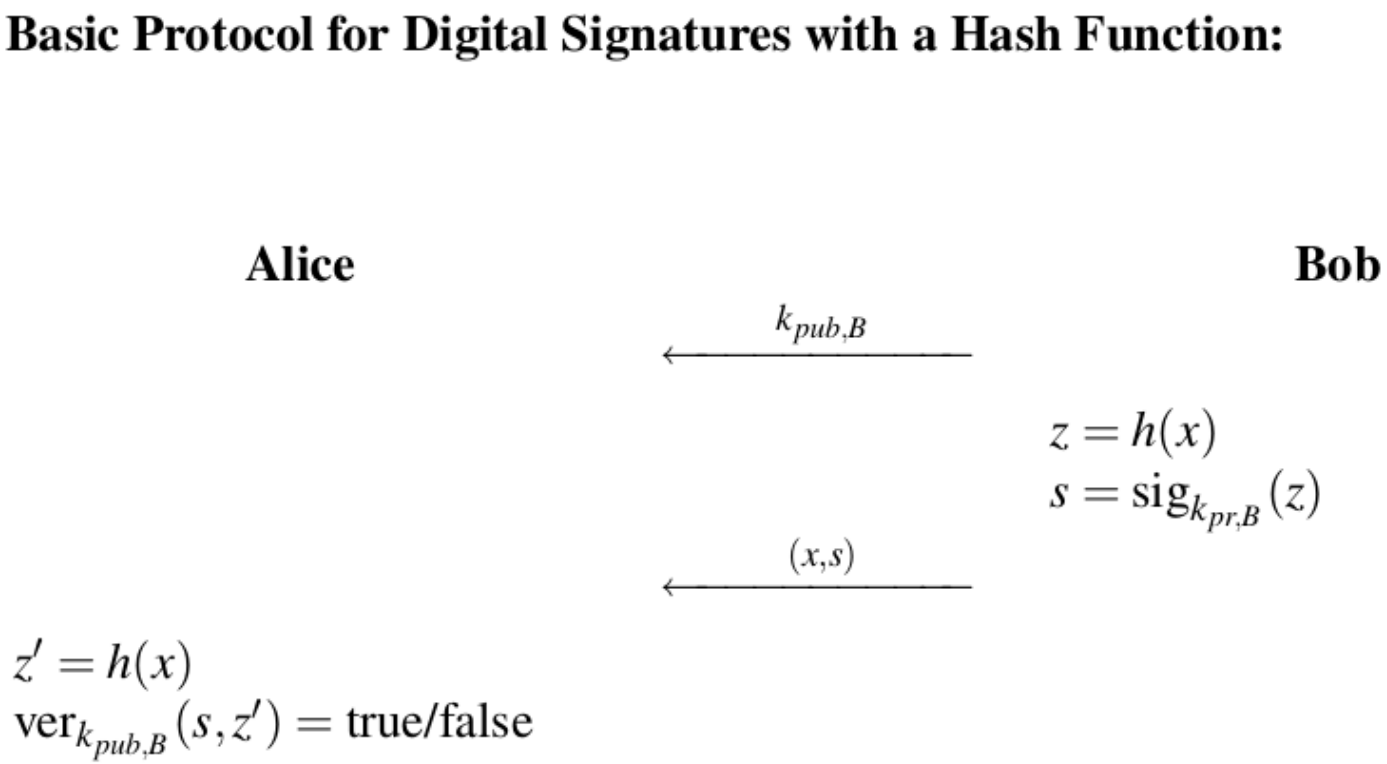


Figure: Basic Protocol for Digital Signature with a Hash Function

Basic Protocol for Digital Signatures with a Hash Function:

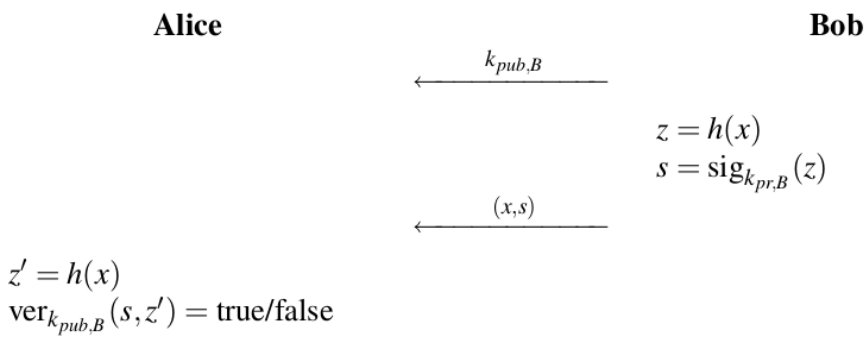


Figure: Basic Protocol for Digital Signature with a Hash Function

- Bob calcula el hash del mensaje de x y firma el valor hash z con su clave privada $k_{pr,B}$.
- En el lado receptor, Alice calcula el valor de hash de la z' de el mensaje recibido x.
- Alice verifica que la firma s con la clave publica de Bob $k_{pub,B}$.
- Se puede observar que tanto en la generación de la firma y la verificación trabajan en el valor hash z en lugar de en el propio mensaje.
- Por lo tanto, el valor de hash representa al mensaje.
- El valor hash se refiere a veces como el resumen del mensaje o la huella digital del mensaje.

Propiedades Deseables

Ahora podemos tener una idea aproximada de la conducta de entrada-salida deseable de funciones hash.

Propiedades Deseables:

- Queremos ser capaces de aplicar una función hash a un mensaje x de cualquier tamaño, y por lo tanto es deseable que la función h es sea computacionalmente eficiente.
- Otra propiedad deseable es que la salida de una función de hash es de longitud fija e independiente de la longitud de entrada.
- Funciones de hash prácticos tienen longitudes de salida entre $128 - 512$ bits.
- Por último, la firma digital ha calcular debe ser muy sensibles a todos los bits de entrada. Eso significa que incluso si hacemos modificaciones menores a la entrada x (cambiamos unos pocos bits del mensaje), la firma digital debe ser muy diferente. Este comportamiento es similar a la de cifrado por bloques.

Propiedades Deseables

Ahora podemos tener una idea aproximada de la conducta de entrada-salida deseable de funciones hash.

Propiedades Deseables:

- Queremos ser capaces de aplicar una función hash a un mensaje x de cualquier tamaño, y por lo tanto es deseable que la función h es sea computacionalmente eficiente.
- Otra propiedad deseable es que la salida de una función de hash es de longitud fija e independiente de la longitud de entrada.
- Funciones de hash prácticos tienen longitudes de salida entre $128 - 512$ bits.
- Por último, la firma digital ha calcular debe ser muy sensibles a todos los bits de entrada. Eso significa que incluso si hacemos modificaciones menores a la entrada x (cambiamos unos pocos bits del mensaje), la firma digital debe ser muy diferente. Este comportamiento es similar a la de cifrado por bloques.

Propiedades Deseables

Ahora podemos tener una idea aproximada de la conducta de entrada-salida deseable de funciones hash.

Propiedades Deseables:

- Queremos ser capaces de aplicar una función hash a un mensaje x de cualquier tamaño, y por lo tanto es deseable que la función h es sea computacionalmente eficiente.
- Otra propiedad deseable es que la salida de una función de hash es de longitud fija e independiente de la longitud de entrada.
- Funciones de hash prácticos tienen longitudes de salida entre $128 - 512$ bits.
- Por último, la firma digital ha calcular debe ser muy sensibles a todos los bits de entrada. Eso significa que incluso si hacemos modificaciones menores a la entrada x (cambiamos unos pocos bits del mensaje), la firma digital debe ser muy diferente. Este comportamiento es similar a la de cifrado por bloques.

Propiedades Deseables

Ahora podemos tener una idea aproximada de la conducta de entrada-salida deseable de funciones hash.

Propiedades Deseables:

- Queremos ser capaces de aplicar una función hash a un mensaje x de cualquier tamaño, y por lo tanto es deseable que la función h es sea computacionalmente eficiente.
- Otra propiedad deseable es que la salida de una función de hash es de longitud fija e independiente de la longitud de entrada.
- Funciones de hash prácticos tienen longitudes de salida entre $128 - 512$ bits.
- Por último, la firma digital ha calcular debe ser muy sensibles a todos los bits de entrada. Eso significa que incluso si hacemos modificaciones menores a la entrada x (cambiamos unos pocos bits del mensaje), la firma digital debe ser muy diferente. Este comportamiento es similar a la de cifrado por bloques.

Propiedades Deseables

Las propiedades que acabamos de describir están simbolizados en la figura.

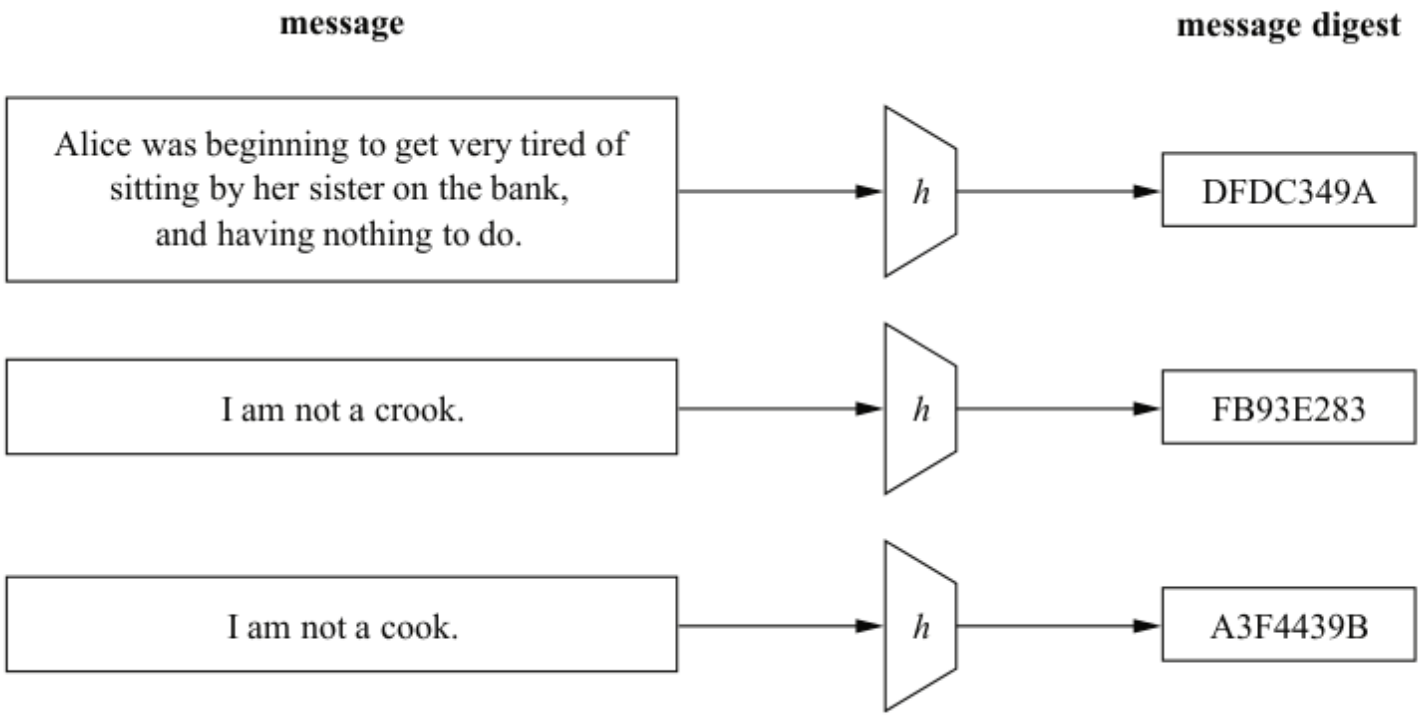


Figure: Conducta de entrada-salida principal de las funciones hash

Requisitos de seguridad de las funciones hash

- Como se mencionó en la introducción, a diferencia de todos los otros algoritmos de cifrado que hemos tratado, las funciones hash no tienen claves.
- La pregunta que nace es si hay algunas propiedades especiales que necesitamos para las función hash sean seguras.
- De hecho, tenemos que preguntarnos si las funciones hash tendrán algun impacto en la seguridad del sistema, ya que no encriptan y no tienen claves.
- Como suele ser el caso en la criptografía, las cosas pueden ser difíciles y hay ataques que utilizan la debilidad de las funciones hash.

Función de hash

Resulta que hay tres características centrales que las funciones hash deben poseer para ser seguras:

1. Resistencia imagen original (o uno-way).
2. Segunda resistencia pre-imagen (o resistencia de colisión débil).
3. Resistencia a la colisión (o resistencia colisión fuerte).

Estas tres propiedades se visualizan en la siguiente figura:

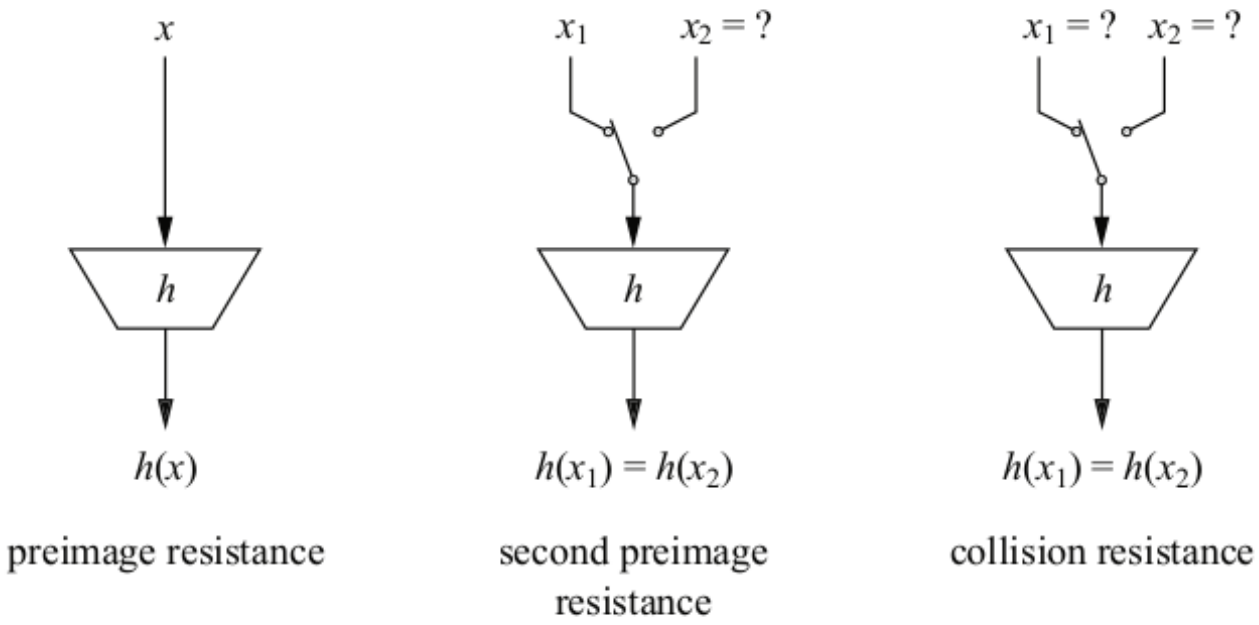


Figure: Las tres propiedades de seguridad de funciones hash

Funciones de Hash

1) Resistencia uno-way
2) Resistencia de colisión débil
3) Resistencia colisión fuerte

- La funciones de hash deben ser *one way*, es decir:
  Dado un y en la imagen (o recorrido) de la función h sea computacionalmente infactible encontrar un valor x en la pre-imagen (o dominio) tal que:

$$h(x) = y$$

- Una función de hash es llamada *collision resistant*
  Si es infactible encontrar dos valores distintos x_1 y x_2 tal que:

$$h(x_1) = h(x_2)$$

- Otra propiedad que deben satisfaces las funciones de hash es el llamado *second preimage resistant*.
  Esta propiedad es que dado un mensaje m debe ser dificil encontrar otro mensaje m_1 tal que $m_1 \neq m$, los valores de hash:

$$h(m_1) = h(m)$$

Por qué one-way function?

- Dada una salida de hash z debe ser computacionalmente imposible encontrar un mensaje de entrada x tal que $z = h(x)$.
- En otras palabras, dada un valor de hash z, no podemos encontrar un mensaje x coincidente.
- Supongamos que Bob está encriptando el mensaje, pero no la firma, es decir, que transmite el par:

$$(e_k(x), sig_{k_{pr},B}(z))$$

- Aquí, $e_k()$ es un cifrado simétrico, por ejemplo, AES, con alguna clave simétrica compartida por Alice y Bob.
- Supongamos que Bob utiliza una firma digital RSA, donde la firma se calcula como:

$$s = sig_{k_{pr},B}(z) \equiv z^d \bmod n$$

- El atacante "Oscar" puede utilizar la clave pública de Bob para calcular

$$s^e \equiv z \bmod n$$

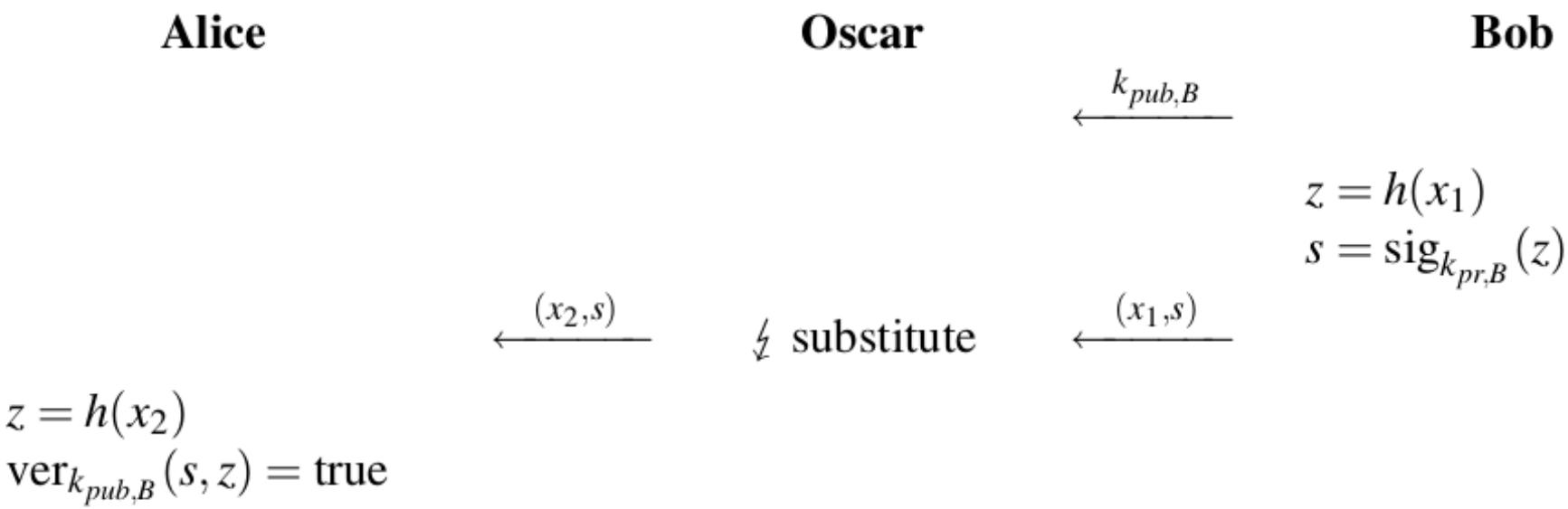
- Si la función hash no es de una *one-way*, Oscar puede ahora calcular el mensaje x ya que $h^{-1}(z) = x$.
- Por lo tanto, el cifrado simétrico de x se elude por la firma, que se escapa del plaintext.

Por qué Resistencia Colisión?

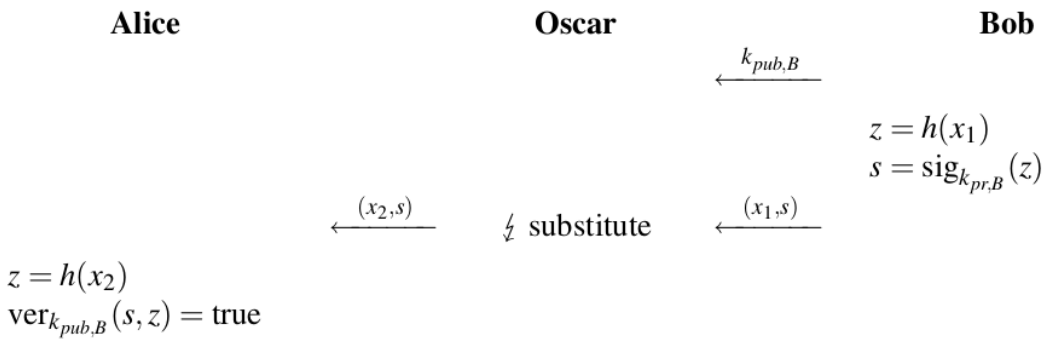
- Para las firmas digitales con hash es esencial que dos mensajes diferentes $x_1 \neq x_2$ no tienen el mismo valor de hash.
- Esto significa que debe ser computacionalmente imposible crear dos mensajes diferentes $x_1 \neq x_2$ con valores hash iguales $z_1 = h(x_1) = h(x_2) = z_2$.
- Se distinguen dos tipos diferentes de tales colisiones:
  - En el primer caso, si tenemos x_1 y tratamos de encontrar x_2. Esto se llama *second preimage resistance* o *weak collision resistance*.
  - El segundo caso se da cuando el atacante es libre de elegir los mensajes x_1 y x_2. Esto se conoce como *strong collision resistance*.

Second preimage resistance

- Es fácil ver por qué *second preimage resistance* es importante para firmas el esquema de hash.
- Supongamos Bob aplica una función de hash y firma un mensaje de x_1.
- Si Oscar "el atacante" es capaz de encontrar un segundo mensaje x_2 tal que $h(x_1) = h(x_2)$, puede ejecutar el siguiente ataque substitución:


Alice
Oscar
Bob
$k_{pub,B}$
$z = h(x_1)$
$s = \mathrm{sig}_{k_{pr,B}}(z)$
(x_2, s)
substitute
(x_1, s)
$z = h(x_2)$
$\mathrm{ver}_{k_{pub,B}}(s, z) = \mathrm{true}$

Colisiones débiles



- Como podemos ver, Alice aceptaría x_2 como mensaje correcto, ya que la verificación le da la declaración, "Verdadera".
- Cómo puede suceder esto? Desde un punto de vista más abstracto, este ataque es posible debido a que tanto la firma (por Bob) y verificación (por Alice) no se producen en el propio mensaje real, sino más bien con la versión de hash de la misma.
- Por lo tanto, si un atacante logra encontrar un segundo mensaje con la misma firma de hash (es decir, la salida del hash), la firma y la verificación son los mismos para este segundo mensaje.

Colisiones débiles

- La pregunta ahora es cómo podemos evitar que Oscar encuentre x_2.
- Idealmente, nos gustaría tener una función hash para el que no existen colisiones débiles.
- Esto es, por desgracia, imposible.
- Dada la salida de cada función hash tiene una longitud fija de bits, por ejemplo n-bits, hay, "solo", 2^n posibles valores de salida de la función de hash.
- Al mismo tiempo, el número de entradas a las funciones de hash es "infinito" de manera que múltiples entradas deben tener el mismo valor de salidad de hash.
- En la práctica, cada valor de salida tiene la misma probabilidad para una entrada al azar, de modo que existen colisiones débiles para todos los valores de salida.

Colisiones débiles

- Dado que existen colisiones débiles en la teoría, la mejor cosa que podemos hacer es asegurar que no se pueden encontrar en la práctica.
- Un función de hash fuerte debe ser diseñada de tal manera que: dada:

$$\text{Dados } x_1 \text{ y } h(x_1) \text{ es imposible construir } x_2 \text{ tal que } h(x_1) = h(x_2).$$

- Esto significa que no hay ataque analítico.
- Sin embargo, Oscar siempre puede elegir al azar los valores x_2, y calcular sus valores hash y comprobar si son iguales a $h(x_1)$. Esto es similar a una búsqueda de clave exhaustiva para un cifrador simétrico "Fuerza Bruta".
- A fin de evitar este ataque. Los computadores de hoy en día, la longitud de salida de $n = 80$ bits es suficiente.
- Sin embargo, veremos en la siguiente sección que los ataques más poderosos que existen nos obligan a usar salidas aún más largas de longitudes en bits de salida de la función de hash.

Strong Collision Resistance

Resistencia de colisión y el Ataque de cumpleaños

- Llamamos una función resistente a las colisiones o resistente a las colisiones fuertes si es computacionalmente imposible encontrar dos mensajes distintos tal que:

$$x_1 \neq x_2 \text{ con } h(x_1) = h(x_2).$$

- Veamos cómo un atacante podría alterar los mensajes para encontrar valores de hash similares.

Strong Collision Resistance

- Él atacante comienza con dos mensajes, por ejemplo:

$$x_1 = \text{Transfers \$ 10 into Oscar's account}$$

$$x_2 = \text{Transfers \$ 10,000 into Oscar's account}$$

- Él atacante altera x_1 y x_2 en lugares "no visibles", por ejemplo, reemplaza los espacios por tabuladores, agrega espacios o signos de retorno al final del mensaje, etc.
- De esta manera, la semántica del mensaje no se ha modificado (por ejemplo, para un banco), pero hay cambios en el valor hash para cada versión del mensaje.
- Oscar continúa hasta que la condición

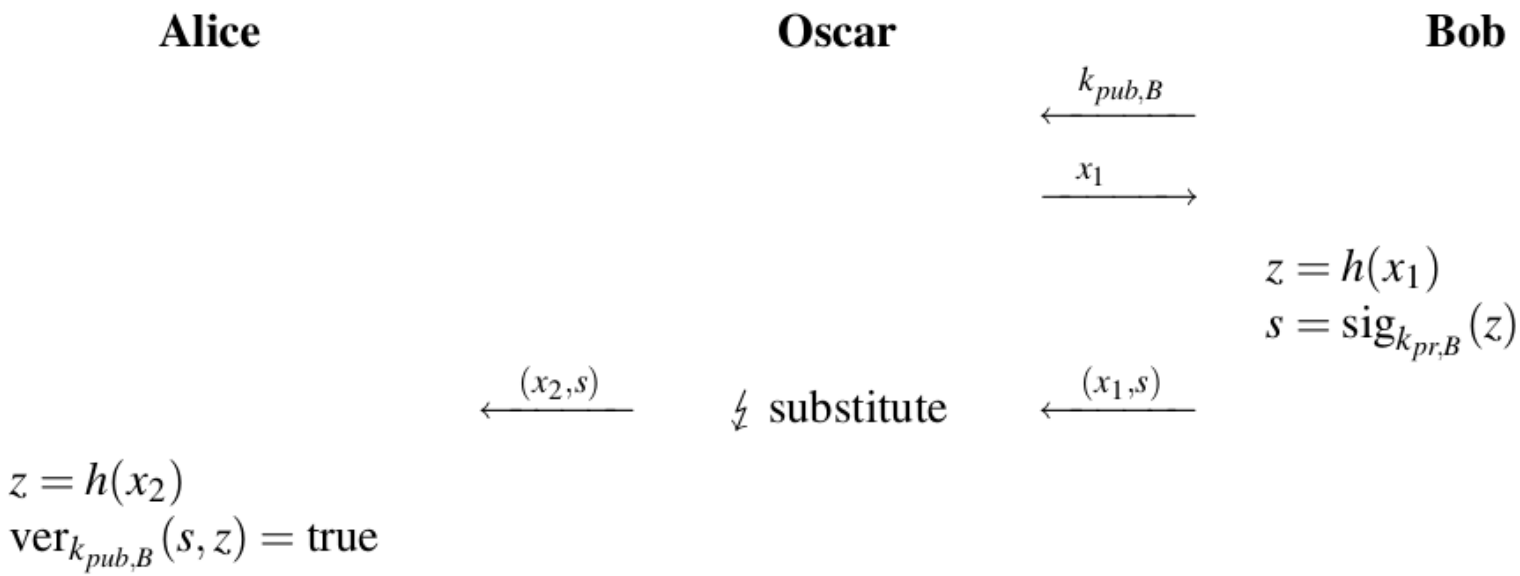
$$h(x_1) = h(x_2)$$

se cumple.

- Se debe observar, que si un atacante tiene, por ejemplo, 64 lugares que puede alterar o no el mensaje, esto da 2^{64} versiones del mismo mensaje con 2^{64} valores hash diferentes.

Strong Collision Resistance

- Con los dos mensajes, el atacante puede realizar el siguiente ataque:



Este ataque supone que Oscar puede engañar a Bob para que firme el mensaje x_1.

Strong Collision Resistance

- Es decir, por supuesto, no es posible en todas las situaciones, pero uno puede imaginar escenarios en los que Oscar puede hacerse pasar por un inocente, por ejemplo, un proveedor de comercio electrónico en Internet, y x_1 es la orden de compra que se genera por Oscar.
- Como vimos anteriormente, siempre existen colisiones.
- La pregunta es qué tan difícil es encontrarlos.

Strong Collision Resistance

- Nuestra primera conjetura es que probablemente difícil encontrar una segunda pre-image, ya que, si la función de hash de salida tiene una longitud de 80-bits, tenemos que comprobar unos 2^{80} mensajes.
- Sin embargo, resulta que un atacante sólo necesita alrededor de 2^{40} mensajes!
- Este es un resultado muy sorprendente que se debe a la paradoja de cumpleaños.
- Resulta que la siguiente pregunta en el mundo real está estrechamente relacionado con la búsqueda colisiones de funciones hash:

¿Cuántas personas se necesitan en una fiesta de tal manera que hay una probabilidad razonable de que al menos dos personas tienen la misma fecha de nacimiento?

Ataque de los Cumpleaños

- Por cumpleaños nos referimos a cualquier de los 365 días del año.
- Nuestra intuición nos puede llevar a suponer que necesitamos alrededor de 183 personas (es decir, alrededor de la mitad del número de días en un año) para que se produzca una colisión.
- Sin embargo, resulta que necesitamos mucha menos personas.
- El enfoque por tramos para resolver este problema consiste en calcular primero la probabilidad de que dos personas que no tienen la misma fecha de nacimiento, es decir, que no tiene colisión de sus cumpleaños.
- Para una persona, la probabilidad de colisión no es 1, lo que es trivial, ya que una sola cumpleaños no puede chocar con cualquier otra persona.
- Para la segunda persona, la probabilidad de ninguna colisión es 364/365, ya que no es sólo un día, el cumpleaños de la primera persona, a chocar con:

Ataque de los Cumpleaños

$$\mathbb{P}(\text{no cumpleaños de una persona}) = \frac{365}{365} = 1$$

$$\mathbb{P}(\text{no colisión de 2 personas}) = \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)$$

$$\mathbb{P}(\text{no colisión de 3 personas}) = \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\left(\frac{363}{365}\right)$$

$$\vdots$$

$$\mathbb{P}(\text{no colisión de } t \text{ personas}) = \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\left(\frac{363}{365}\right)\cdots\left(\frac{365-(t-1)}{365}\right)$$

Ataque de los Cumpleaños

- Volvamos a nuestra pregunta inicial:
  Cuántas personas son necesarias para tienen una probabilidad del 50% de los dos cumpleaños en colisión?
- Sorprendentemente, después de la ecuaciones anteriores "sólo requiere 23" personas para obtener una probabilidad de aproximadamente el 0.5 por una colisión de cumpleaños, ya que:

$$\begin{aligned}\mathbb{P}(\text{a lo menos una colisión}) &= 1 - \mathbb{P}(\text{no colisión}) \\ &= 1 - (\frac{365}{365})(\frac{364}{365})(\frac{363}{365})\cdots(\frac{365-(23-1)}{365}), \\ &= 0.507 \approx 50\%.\end{aligned}$$

Colisiones para las funciones de hash $h()$

- Encontrar colisiones en las funciones de hash es el mismo problema de encontrar colisiones en la paradoja del cumpleaños.
- Para una función de hash de salida un string de tamaño n tenemos 2^n posibles valores.
- De hecho, el tamaño de n es el parametro de seguridad crucial para la función de hash.

- La pregunta ¿Cuantos mensajes $(x_1, x_2, \ldots, x_t)$ necesita un atacante digamos Oscar hasta encontrar una colisión?

$$h(x_i) = h(x_j) \text{ para algún } x_i \text{ y } x_j$$

- La probabilidad es :

$$\begin{aligned}\mathbb{P}(\text{No Colisión}) &= (1 - \frac{1}{2^n})(1 - \frac{2}{2^n}) \cdots (1 - \frac{t-1}{2^n}) \\ &= \prod_{i=1}^{t-1}(1 - \frac{i}{2^n})\end{aligned}$$

Recordar de las series de Taylor que:

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3} + \cdots \text{ para } x \ll 1.$$

Entonces $e^{-x} \approx 1 - x$, ya que $\frac{i}{2^i} \ll 1$.

Podemos aproximar la probabilidad

$$\begin{aligned}\mathbb{P}(\text{no colisión}) &\approx \prod_{i=1}^{t-1} e^{-\frac{i}{2^n}} \\ &\approx e^{-\frac{1+2+3+\cdots+t-1}{2^n}}\end{aligned}$$

Sabemos que $1+2+\cdots+t-1=\frac{t(t-1)}{2}$. Lo que implica que

$$\mathbb{P}(\text{no colisión}) \approx e^{-\frac{t(t-1)}{2\cdot 2^n}},$$

Nuestro objetivo es averiguar cuntos mensajes $(x_1, x_2, \ldots, x_t)$ se necesitan para encontrar una colisión en la función de hash h.

Por lo tanto, se resuelve la ecuación en función de t. Si denotamos la probabilidad de al menos una colisión por λ, entonces:

$$\lambda = 1 - \mathbb{P}(\text{no colisión})$$

entonces:

$$\lambda \approx 1 - e^{-\frac{t(t-1)}{2^{n+1}}}$$

$$\ln(1-\lambda) \approx -\frac{t(t-1)}{2^{n+1}}$$

$$t(t-1) \approx 2^{n+1} \ln(\frac{1}{1-\lambda}).$$

En la practica $t \gg 1$ se cumple que $t^2 \approx t(t-1)$ entonces:

$$t \approx \sqrt{2^{n+1} \ln(\frac{1}{1-\lambda})}$$

$$t \approx 2^{(n+1)/2} \sqrt{\ln(\frac{1}{1-\lambda})}$$

La ecuación

$$t \approx 2^{(n+1)/2}\sqrt{\ln(\frac{1}{1-\lambda})}$$

es extremadamente importante:

Describe la relación entre el número de mensajes t necesario para una colisión en una función de hash de longitud de salida n y la probabilidad de colisión λ.

- La consecuencia más importante del ataque de los cumpleaños es que el número de mensajes necesarios para encontrar una colisión en una función de hash es aproximadamente igual a la raíz cuadrada del número de posibles valores de salida, es decir, aproximadamente

$$\sqrt{2^n} = 2^{n/2}$$

- Por lo tanto, para un nivel de seguridad de x-bits, la función hash tiene que tener una longitud de salida de $2x$-bits.
- A modo de ejemplo, supongamos que desee encontrar una colisión de una función hash hipotética con salida de 80 bits. Para una probabilidad de éxito del 50%, esperamos que el hash cerca:

$$t = 2^{81/2}\sqrt{\ln(1/1-0.5)} \approx 2^{40.2}$$

valores de entrada.

- Cálculando alrededor de 2^{40} valores hash y comprobando las colisiones se puede hacer con los ordenadores portátiles actuales!.
- Por esta razón, todas las funciones de hash tienen una longitud de salida de al menos 128-bits, donde la mayoría de los modernos son mucho más largos.

- En la siguiente Tabla, se muestran el número de cálculos necesarios para hash de una colisión cumpleaños-paradoja para longitudes de salida que se encuentran en las funciones de hash actuales.

Table: Número de valores hash necesarios para la colisión de diferentes longitudes de salida de la función de hash y de dos probabilidades de colisión diferentes

	Longitud de Salida de Hash				
λ	128 bit	160 bit	256 bit	384 bit	512 bit
0.5	2^{65}	2^{81}	2^{129}	2^{193}	2^{257}
0.9	2^{67}	2^{82}	2^{130}	2^{194}	2^{258}

- Curiosamente, la probabilidad deseada de una colisión no influye en la complejidad ataque, como se evidencia por la pequeña diferencia entre las probabilidades de éxito $\lambda = 0.5$ y $\lambda = 0.9$.

- Cabe destacar que el ataque de cumpleaños es un ataque genérico. Esto significa que es aplicable contra cualquier función hash.
- Por otro lado, no se garantiza que sea el ataque más poderoso disponible para un determinado función hash.
- Existen muchas aplicaciones para las funciones hash, por ejemplo: El almacenamiento de contraseñas, que sólo requieren resistencia preimagen. Por lo tanto, una función hash con un tiempo relativamente corto de salida, por ejemplo 80 bits, podría ser suficiente, ya que los ataques de colisión no suponen una amenaza.

Propiedades de la Funciones de Hash

En resumen todas las propiedades importantes de las funciones hash $h(x)$.

Tamaño del mensaje arbitraria $h(x)$ se puede aplicar a los mensajes x de cualquier tamaño.

Salida fija longitud de $h(x)$ produce un valor hash de z de longitud fija.

Eficiencia $h(x)$ es relativamente fácil de calcular.

Resistencia preimagen Para una salida z dada, es imposible encontrar cualquier de entrada x tal que $h(x) = z$, es decir, $h(x)$ es una *one-way function*.

Segunda resistencia preimagen Dado x_1, y por lo tanto $h(x_1)$, es computacionalmente infactible encontrar un mensaje x_2 tal que $h(x_1) = h(x_2)$.

Resistencia Collision Es computacionalmente imposible encontrar pares $x_1 = x_2$ tal que $h(x_1) = h(x_2)$.

Observación

Las tres primeras propiedades son los requisitos prácticos, mientras que las tres últimas se relacionan a la seguridad de las funciones de hash

Función de hash SHA-1

- El algoritmo de hash seguro (SHA-1) (En la actualidad NIST-2012 estandarizo el SHA-3) es la función de hash más ampliamente utilizada.
- Aunque se han propuesto nuevos ataques contra el algoritmo, es muy instructivo examinar sus detalles porque la version más fuerte de la familia SHA-2 muestran una estructura interna muy similar.
- SHA-1 se basa en una construcción Merkle-Damgard, como puede verse en la siguiente figura.
- Una interpretación interesante del algoritmo SHA-1 es que la función de compresión trabaja como un cifrado de bloques, donde la entrada es el valor hash anterior H_{i-1} y la clave está formado por el bloque de mensaje x_i.

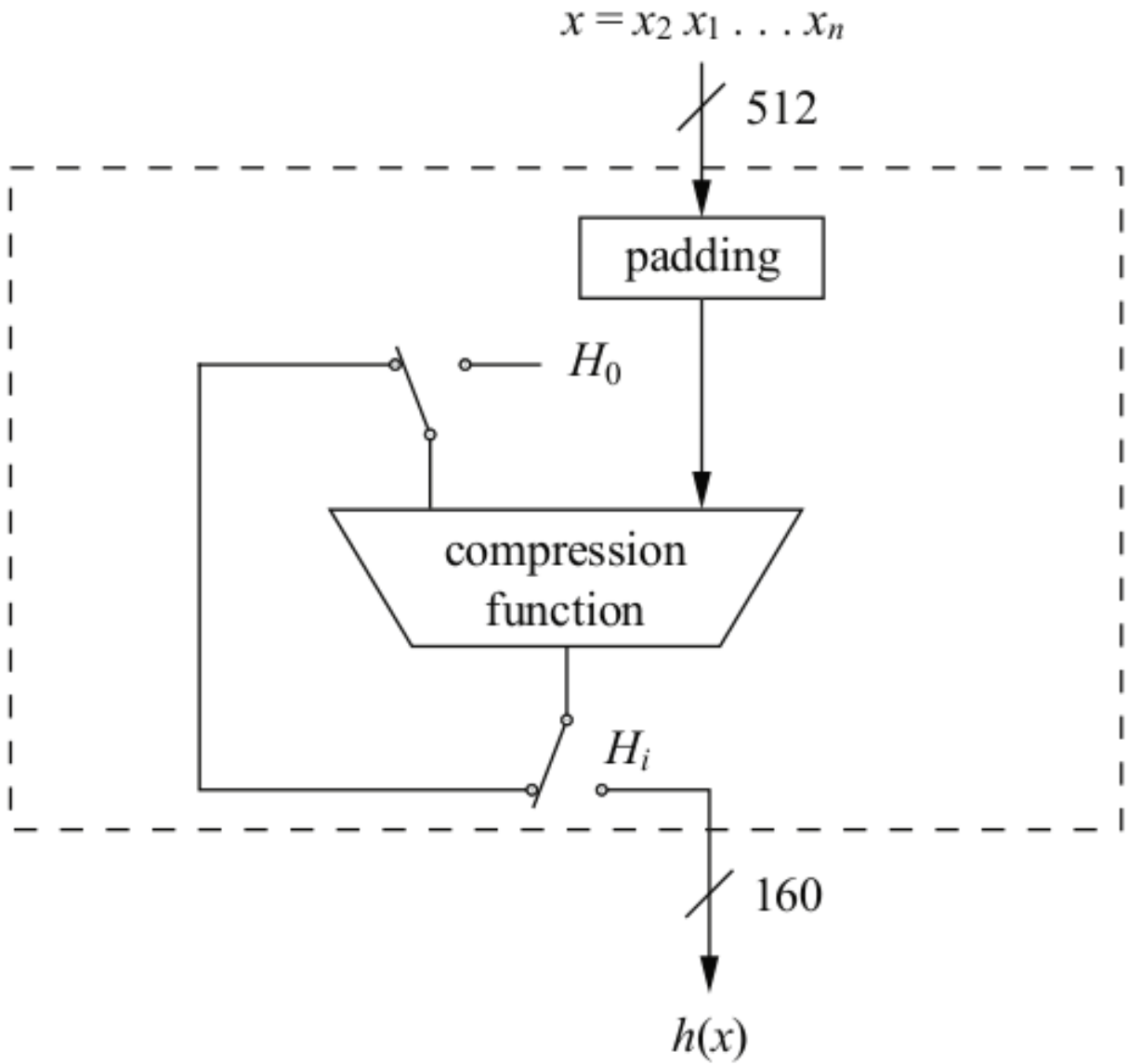


Figure: Diagrama de alto nivel del SHA-1

Función de hash SHA-1

- Como veremos más adelante, las rondas de SHA-1 son muy similar a un bloque de cifrado Feistel.
- SHA-1 produce una salida de 160-bits de un mensaje con una longitud máxima de 2^{64}-bits.
- Antes de que el cálculo del valor hash, el algoritmo tiene que preprocesar el mensaje.
- Durante el cálculo, la función de compresión procesa el mensaje de largo 512-bits
- La función de compresión consta de 80-rondas que se dividen en cuatro etapas de 20-vueltas cada una.

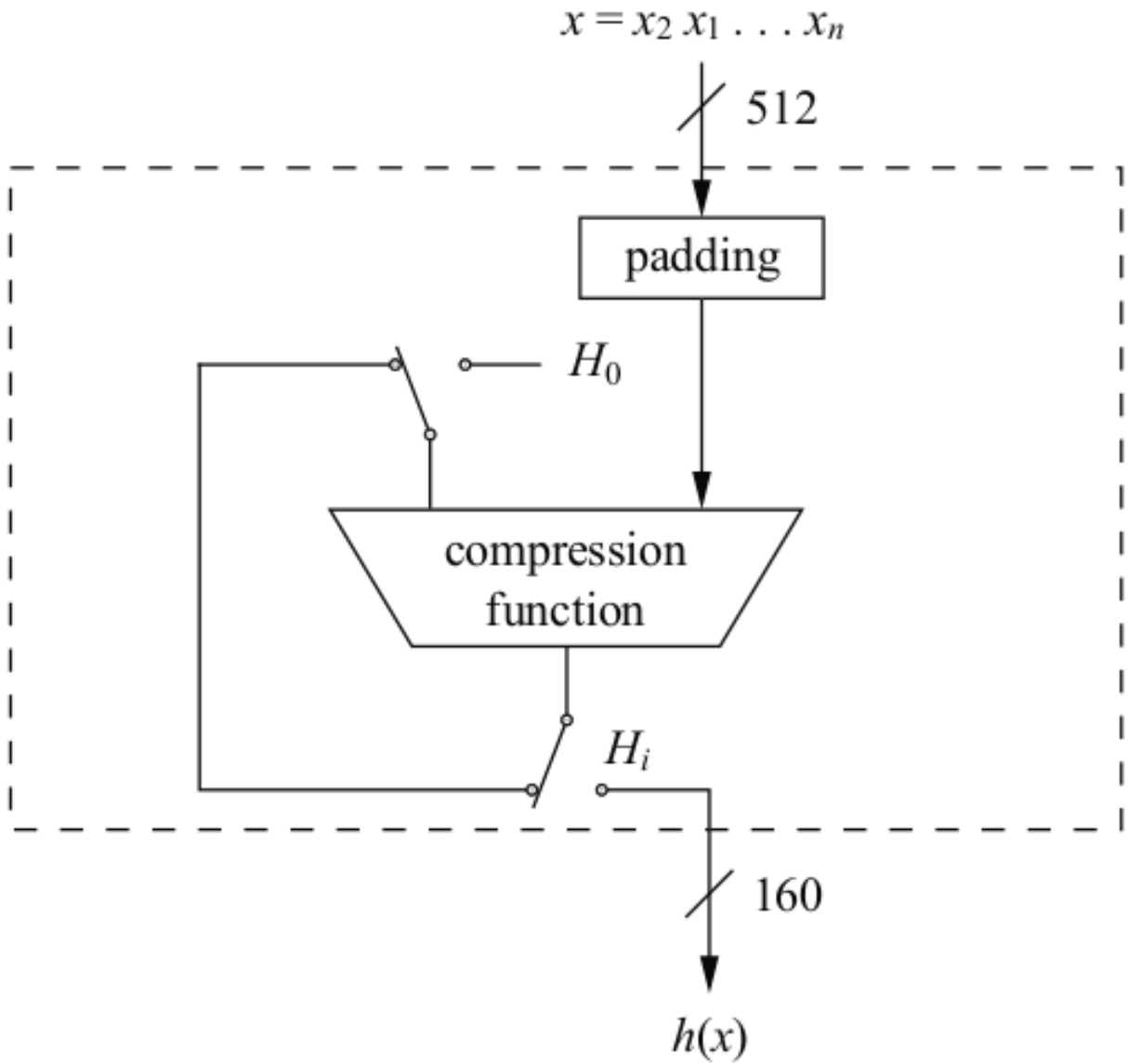


Figure: Diagrama de alto nivel del SHA-1

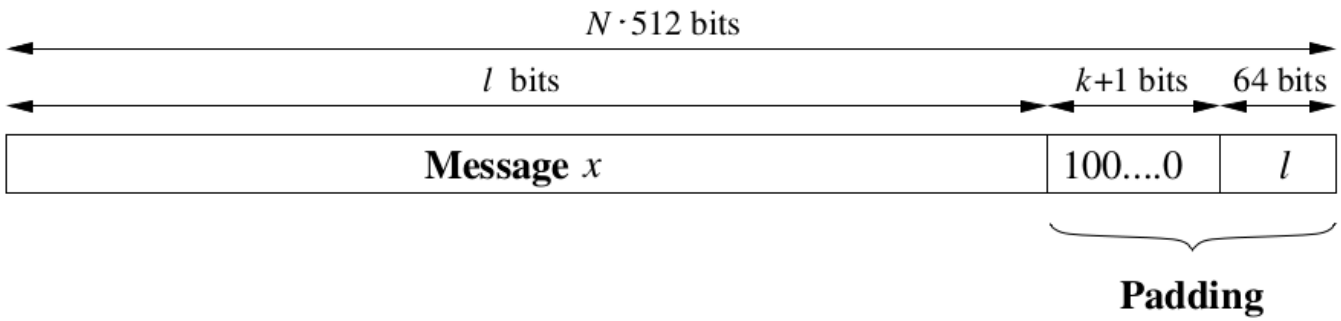
Preprocesamiento

- Antes de calcular del valor hash, el mensaje x tiene que ser rellenado hasta el ajuste a un tamaño de un múltiplo de 512-bits.
- Para el procesamiento interno, el mensaje rellenado se dividirse en bloques.
- También, el valor inicial H_0 se establece como una constante predefinida.

Relleno: Supongamos que tenemos un mensaje x con una longitud de l-bits. Para obtener una tamaño total del mensaje de un múltiplo de 512-bits, que anexar un solo "1" seguido por k bits cero y la representación de 64-bits binarios de l. Por consiguiente, el número de ceros necesarios k viene dada por:

$$k \equiv 512 - 64 - 1 - l \tag{1}$$

$$= 448 - (l + 1) \bmod 512 \tag{2}$$



Dividiendo el Mensaje Relleno

- Antes de la aplicación de la función de compresión, hay que dividir el mensaje en bloques de 512-bits

$$x_1, x_2, \ldots, x_n$$

- Cada bloque de 512-bits debe ser subdividido en 16 palabras de tamaño de 32-bits.
- Por ejemplo, el bloque i-ésimo de el mensaje x se divide en:

$$x_i = (x_i^{(0)} x_i^{(1)} \cdots x_i^{(15)})$$

donde los $x_i^{(k)}$ son palabras de tamaño de 32 bits.

Valor inicial H_0

Un buffer de 160-bits se utiliza para mantener el valor de hash inicial (H_0) para la primera iteración. Las cinco palabras de 32-bits se fijan y se dan en notación hexadecimal como:

$$A = H_0^{(0)} = 67452301$$
$$B = H_0^{(1)} = EFCDAB89$$
$$C = H_0^{(2)} = 98BADCFE$$
$$D = H_0^{(3)} = 10325476$$
$$E = H_0^{(4)} = C3D21EF0$$

Calculo de Hash

Cada mensaje de bloqueo x_i se procesa en cuatro etapas con 20 rondas cada uno:

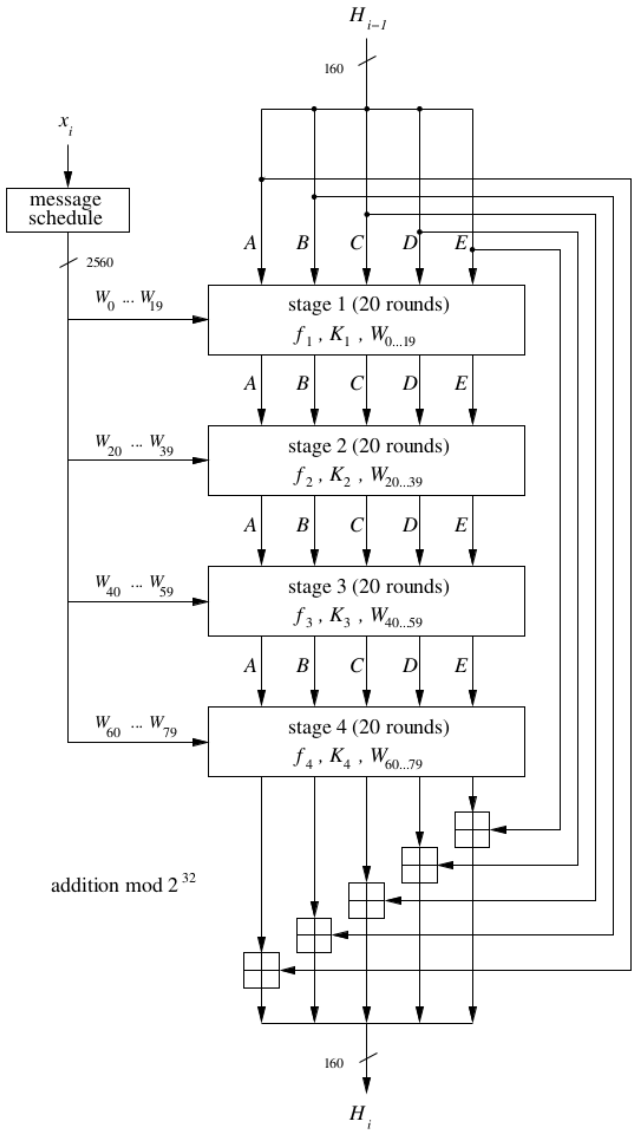


Figure: Ochenta rondas de la función de compresión de SHA-1

Calculo de Hash

El algoritmo utiliza:

- Un mensaje schedule que calcula una palabra de 32-bits $W_0, W_1, \ldots, W_{79}$ para cada uno de los 80-rondas.
- Las palabras W_j se derivan del bloque de mensaje de 512-bits de la siguiente manera:

$$W_j = \begin{cases} x_i^{(j)} & \text{si } 0 \leq j \leq 15, \\ \\ (W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-3})_{\lll 1} & \text{si } 16 \leq j \leq 79, \end{cases}$$

  donde $X_{\lll n}$ indica un left shift circular de la palabra X en n posiciones de bit.
- cinco registros de trabajo de tamaño de 32 bits: A, B, C, D, E.
- Un valor hash H_i consta de cinco palabras de 32-bits $H_i^{(0)}$, $H_i^{(1)}$, $H_i^{(2)}$, $H_i^{(3)}$, $H_i^{(4)}$. En el principio, el valor hash tiene el H_0 valor inicial, que se sustituye por un nuevo valor hash una vez que el procesamiento de cada bloque de mensaje único. El final de H_n valor hash es igual a la salida de $h(x)$ de SHA-1.

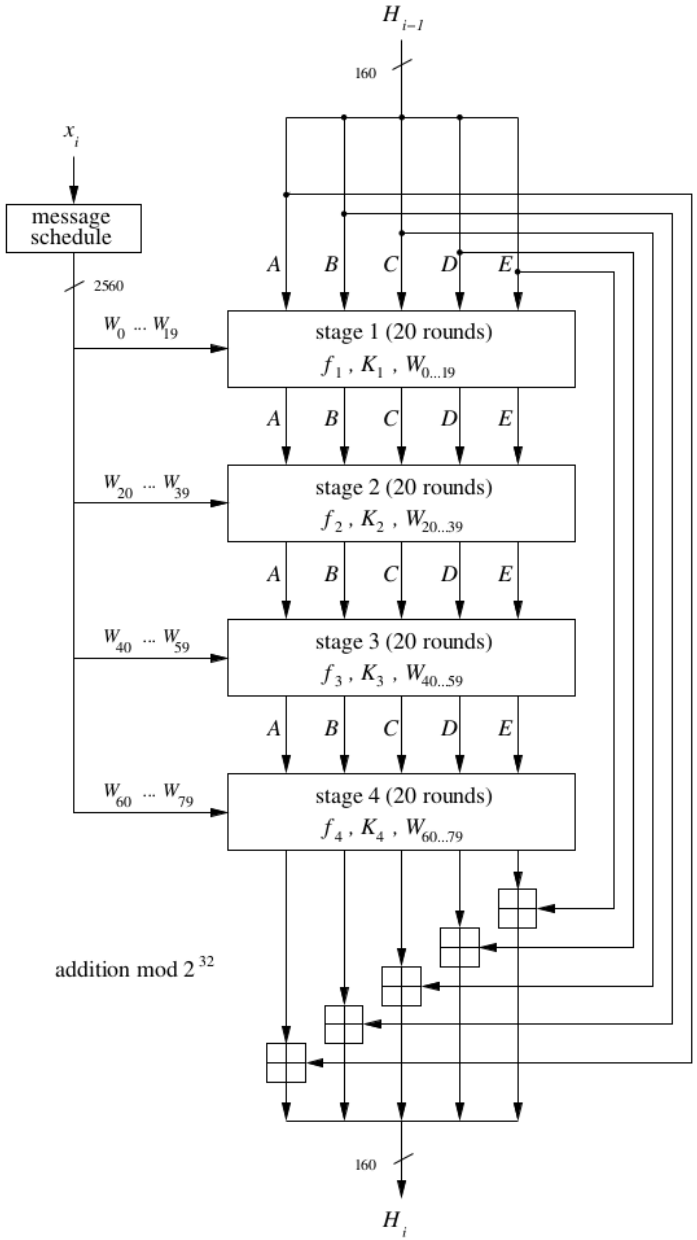
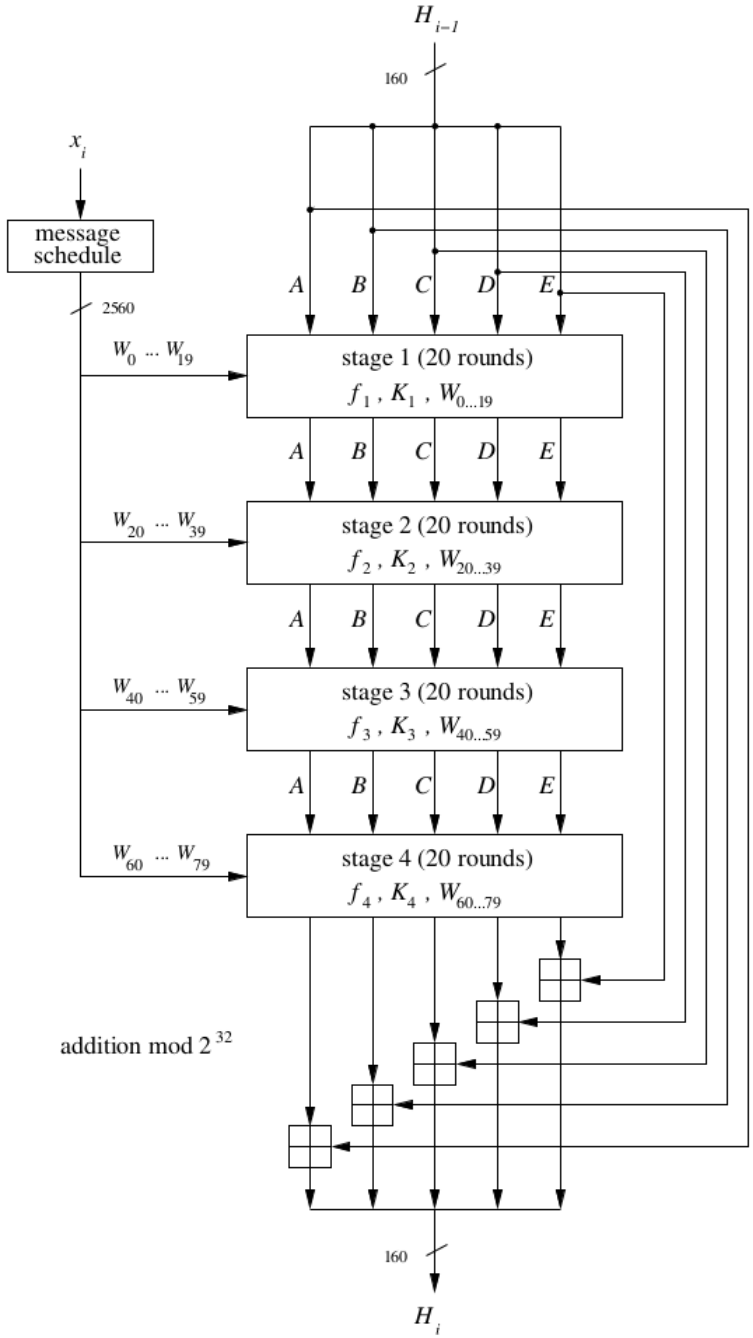


Figure: Ochenta rondas de la función de compresión de SHA-1

- Las cuatro etapas del SHA-1 tienen una estructura similar, pero utilizan diferentes funciones internas f_t y constantes K_t, where $1 \leq t \leq 4$.
- Cada etapa se compone de 20 rondas, donde partes del mensaje de bloque son procesados por la función f_t junto con algún estado dependiente de la constante K_t.
- La salida después de 80-rondas es sumanda la entrada H_{i-1} modulo 2^{32}.



- La operación dentro de la ronda j $(0 \leq j \leq 15)$ en la etapa t $(1 \leq t \leq 4)$ está dada por:

$$A,\ B,\ C,\ D,\ E = (E + f_t(B, C, D) + (A)_{\lll 5} + W_j + K_t),\ A,\ (B)_{\lll 30},\ C,\ D$$

Donde $\boxplus$ es una adición mod 2^{32} y se representa en la siguiente figura.

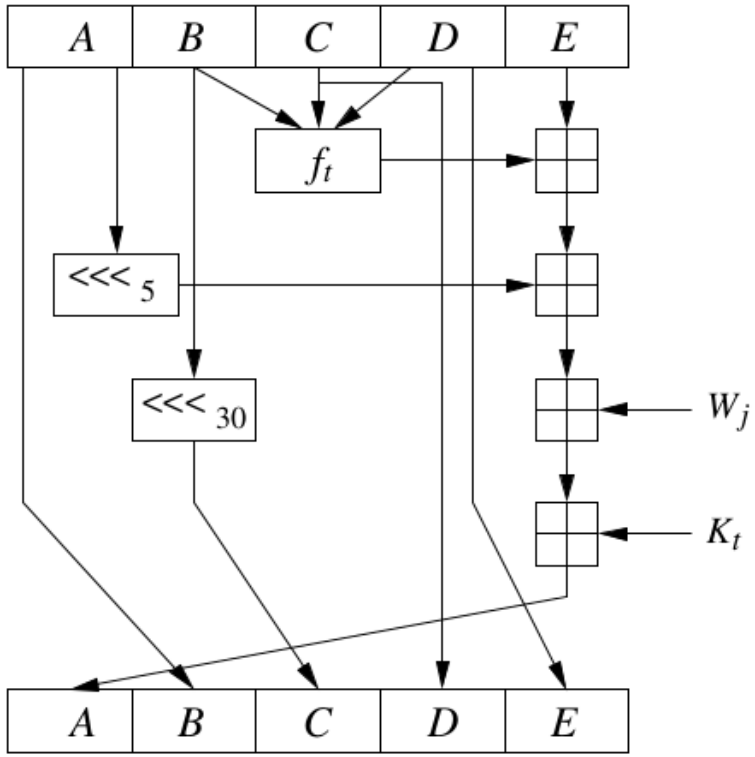


Figure: Ronda j en la etapa t de SHA-1

- Las funciones internas f_t y las constantes de cambio K_t cambian:
  Dependiendo de la etapa de acuerdo con la siguiente Tabla, es decir, cada 20 rondas una nueva función y una nueva constante se están utilizando.
- La función sólo utiliza operaciones bit a bit booleanas, AND ($\wedge$), OR ($\vee$), NOT (barra superior) y XOR.
- Estas operación se aplican a las variables de 32-bits y son muy rápidos para poner en práctica en los ordenadores modernos.

Stage t	Round j	Constant K_t	Function f_t
1	$0 \ldots 9$	$K_1 = 5A827999$	$f_1(B, C, D) = (B \wedge C) \vee (\bar{B} \wedge D)$
2	$20 \ldots 39$	$K_2 = 6ED9EBA1$	$f_2(B, C, D) = B \oplus C \oplus D$
3	$40 \ldots 59$	$K_3 = 8F1BBCDC$	$f_3(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
4	$60 \ldots 79$	$K_4 = CA62C1D6$	$f_4(B, C, D) = B \oplus C \oplus D$

Table: Funciones y Constantes Rondas para el SHA

- Una ronda de SHA-1 como se vio tiene cierta semejanza con la ronda de una red de Feistel.
- Tales estructuras se denominan a veces de redes Feistel generalizado.
- Redes de Feistel se caracterizan generalmente por el hecho de la primera parte de la entrada se copia directamente a la salida.
- La segunda parte de la entrada es encriptada usando la primera parte, donde la primera parte se envía a través de alguna función, por ejemplo, la función f en el caso de DES.
- En la rondas del SHA-1, las entradas A, B, C y D son entregadas a la salida sin cambio (A, C, D), o sólo cambios mínimos (rotación de B).
- Sin embargo, la palabra de entrada E es "encriptada" mediante la adición de valores derivados de la otras cuatro palabras de entrada. El valor del mensaje-derivado W_i y la ronda constante, juega un rol de las subclaves.